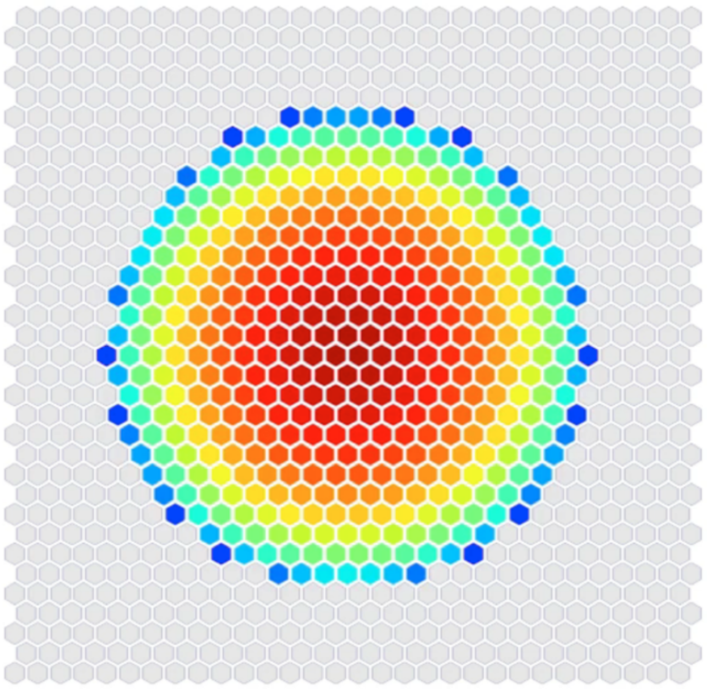


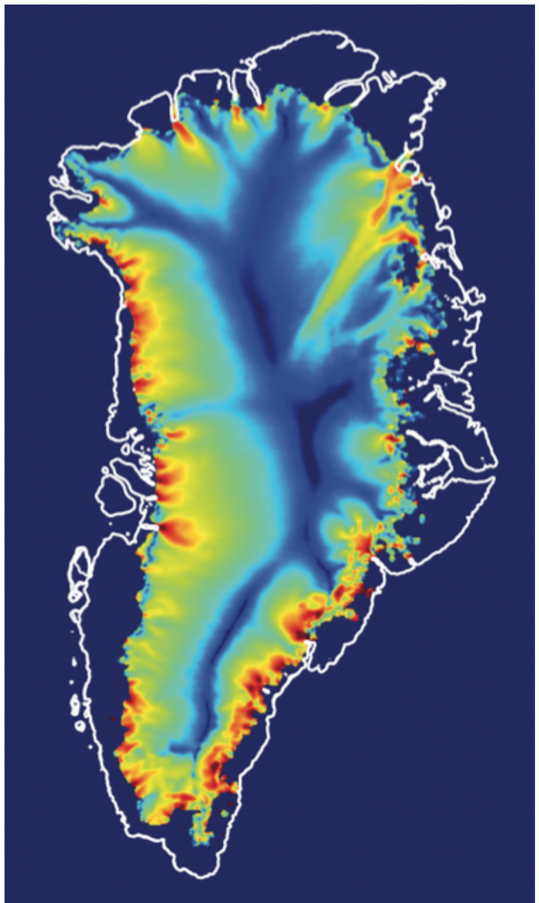
MPAS-Land Ice Model User's Guide

Version: 2.0

Climate, Ocean, Sea-Ice Modeling Team

Los Alamos National Laboratory

September 18, 2013



Foreword

The Model for Prediction Across Scales-Land Ice (MPAS-Land Ice) is an unstructured-mesh land ice model (ice sheets or glaciers) capable of using enhanced horizontal resolution in selected regions of the land ice domain. This allows researchers to perform high-resolution regional simulations at a lower computational cost, while providing realistic ice flow from the low-resolution regions. Model domains may be spherical or on Cartesian domains. MPAS-Land-Ice is being designed for large-scale, hi-resolution simulations of ice sheet dynamics, using a combination of Finite Difference, Finite Volume, and Finite Element Methods on variable resolution meshes. MPAS-Land-Ice will initially be released with support for standard test cases used in verifying model performance. Eventually, support for standard ice sheet configurations will also be included (e.g., stand-alone Greenland and Antarctica simulations).

Prototype dynamcial cores for MPAS-Land-Ice have shown good agreement with manufactured solutions and standard test cases, and have been used in land ice evolution experiments aimed at informing the IPCC AR5 on the potential for future sea-level rise from ice sheets (e.g., Ice2Sea international assessment project (Shannon et al., 2013; Edwards et al., 2013)). Two dynamical cores are currently under development for implementation within MPAS-Land-Ice. These include a 1st-order accurate approximation to the momentum balance equations, a prototype of which has been described by Perego et al. (2012), and a "full" Stokes momentum balance, described in Leng et al. (2012).

MPAS-Land Ice is one component within the MPAS framework of climate models that is developed in cooperation between Los Alamos National Laboratory (LANL) and the National Center for Atmospheric Research (NCAR). Functionality that is required by all cores, such as i/o, time management, block decomposition, etc, is developed collaboratively, and this code is shared across cores within the same repository. Each core then solves its own differential equations and physical parameterizations within this framework. This user's guide reflects the spirit of this collaborative process, where Part I, "The MPAS Framework", applies to all cores, and the remaining parts apply to MPAS-Land Ice.

Here we would normally describe the new features of this version. For the initial release, we will simply review the major features of the basic MPAS-Land Ice model. We employ a finite-volume discretization of the ice continuity equation using a C-grid staggering in the horizontal. The vertical coordinate is sigma. The time-stepping method is Forward Euler (explicit). Ice advection is performed by first-order upwinding. No tracer advection is available at present. In the initial release velocity can only be solved using the Shallow Ice Approximation.

A history of past releases of the Land Ice core within the MPAS version numbering scheme is as follows:

version	date	description
2.0.0	November 15, 2013	Initial public release of Land Ice core (SIA velocity solver only)

Information about MPAS-Land Ice, including the most recent code, user's guide, and test cases, may be found at http://mpas-dev.github.com. This user's guide refers to version 2.0.

Contributors to this guide:
Matt Hoffman, Stephen Price
Additional contributors to MPAS Framework sections:
Michael Duda, Douglas Jacobsen

Funding for the development of MPAS-Land Ice was provided by the United States Department of Energy, Office of Science.

Contents

Chapter 1

MPAS-Land Ice Quick Start Guide

This chapter provides MPAS-Land Ice users with a quick start description of how to build and run the model. It is meant merely as a brief overview of the process, while the more detailed descriptions of each step are provided in later sections.

In general, the build process follows the following steps.

1. Build MPI Layer (OpenMPI, MVAPICH2, etc.)
2. Build serial NetCDF library (v3.6.3, v4.1.3, etc.)
3. Build Parallel-NetCDF library (v1.2.1, v1.3.0, etc.)
4. Build Parallel I/O library (v1.4.1, v1.6.1, etc.)
5. (Optional) Build METIS library and executables (v4.0, v5.0.2, etc.)
6. Checkout MPAS-Land Ice from repository
7. Build Land Ice core

After step 7, an executable should be created called landice_model.exe. Once the executable is built, one can begin the run process as follows:

1. Create run directory.
2. Copy executable to run directory.
3. Copy namelist.input into run directory.
4. (Optional) Copy input and graph files into run directory.
5. Edit namelist.input to have the proper parameters.
   If step 4 was skipped, ensure paths to input and graph files are appropriately set.
6. (Optional) Create graph files, using METIS executable (pmetis or gpmetis depending on version).
   A graph file is required for each processor count you want to use.
7. Run MPAS-Land Ice.
8. Visualize output file, and perform analysis.

Part I

The MPAS Framework

Chapter 2

Building MPAS

2.1 Prequisites

To build MPAS, compatible C and Fortran compilers are required. Additionally, the MPAS software relies on the PIO parallel I/O library to read and write model fields, and the PIO library requires the standard netCDF library as well as the parallel-netCDF library from Argonne National Labs. All libraries must be compiled with the same compilers that will be used to build MPAS. Section 2.2 summarizes the basic procedure of installing the required I/O libraries for MPAS.

In order for the MPAS makefiles to find the PIO, parallel-netCDF, and netCDF include files and libraries, the environment variables `PIO`, `PNETCDF`, and `NETCDF` should be set to the root installation directories of the PIO, parallel-netCDF, and netCDF installations, respectively. Newer versions of the netCDF library use a separate Fortran interface library; the top-level MPAS Makefile attempts to add `-lnetcdff` to the linker flags, but some linkers require that `-lnetcdff` appear before `-lnetcdf`, in which case `-lnetcdff` will need to be manually added just before `-lnetcdf` in the specification of `LIBS` in the top-level Makefile.

An MPI installation such as MPICH or OpenMPI is also required, and there is no option to build a serial version of the MPAS executables. There is currently no support for shared-memory parallelism with OpenMP within the MPAS framework.

2.2 Compiling I/O Libraries

NOTE: It's important to note the MPAS Developers are not responsible for any of the libraries that are used within MPAS. Support for specific libraries should be taken up with the respective developer groups.

Although most recent versions of the I/O libraries should work, the most tested versions of these libraries are: netCDF 4.1.3, parallel-netCDF 1.3.1, and PIO 1.4.1. The netCDF and parallel-netCDF libraries must be installed before building PIO library.

All commands are presented for csh, and will not work if pasted into another shell. Please translate them to the appropraite commands in your shell.

2.2.1 netCDF

Version 4.1.3 of the netCDF library may be downloaded from `http://www.unidata.ucar.edu/downloads/netcdf/netcdf-4_1_3/index.jsp`. Assuming the gfortran and gcc compilers will be used, the following shell commands are generally sufficient to install netCDF.

```
   > setenv FC gfortran
   > setenv F77 gfortran
   > setenv F90 gfortran
   > setenv CC gcc
   > ./configure --prefix=XXXXX --disable-dap --disable-netcdf-4 --disable-cxx
--disable-shared --enable-fortran
   > make all check
   > make install
```

Here, `XXXXX` should be replaced with the directory that will serve as the root installation directory for netCDF. *Before proceeding to compile PIO the* `NETCDF_PATH` *environment variable should be set to the netCDF root installation directory.*

Certain compilers require addition flags in the CPPFLAGS environment variable. Please refer to the netCDF installation instructions for these flags.

2.2.2 parallel-netCDF

Version 1.3.1 of the parallel-netCDF library may be downloaded from https://trac.mcs.anl.gov/projects/parallel-netcdf/wiki/Download. Assuming the gfortran and gcc compilers will be used, the following shell commands are generally sufficient to install parallel-netCDF.

```
   > setenv MPIF90 mpif90
   > setenv MPIF77 mpif90
   > setenv MPICC mpicc
   > ./configure --prefix=XXXXX
   > make
   > make install
```

Here, `XXXXX` should be replaced with the directory that will serve as the root installation directory for parallel-netCDF. *Before proceeding to compile PIO the* `PNETCDF_PATH` *environment variable should be set to the parallel-netCDF root installation directory.*

2.2.3 PIO

Instructions for building PIO can be found at http://www.cesm.ucar.edu/models/pio/. Please refer to these instructions for building PIO.

After PIO is built, and installed the PIO enviroment variable needs to be defined to point at the directory PIO is installed into. Older versions of PIO cannot be installed, and the PIO environment variable needs to be set to the directory where PIO was built instead.

2.3 Compiling MPAS

Before compiling MPAS, the `NETCDF`***,*** `PNETCDF`***, and*** `PIO` ***environment variables must be set to the library installation directories as described in the previous section. A*** `CORE` ***variable also needs to either be defined or passed in during the make process. If*** `CORE` ***is not specified, the build process will fail.***

The MPAS code uses only the 'make' utility for compilation. Rather than employing a separate configuration step before building the code, all information about compilers, compiler flags, etc., is contained in the top-level `Makefile`; each supported combination of compilers (i.e., a configuration) is included in the `Makefile` as a separate make target, and the user selects among these configurations by running `make` with the name of a build target specified on the command-line, e.g.,

```
> make gfortran
```

to build the code using the GNU Fortran and C compilers. Some of the available targets are listed in the table below, and additional targets can be added by simply editing the `Makefile` in the top-level directory.

Target	Fortran compiler	C compiler	MPI wrappers
`xlf`	xlf90	xlc	mpxlf90 / mpcc
`pgi`	pgf90	pgcc	mpif90 / mpicc
`ifort`	ifort	gcc	mpif90 / mpicc
`gfortran`	gfortran	gcc	mpif90 / mpicc
`g95`	g95	gcc	mpif90 / mpicc

In order to get a more complete and up-to-date list of available tagets, one can use the following command within the top-level of MPAS. **NOTE:** This command is known to not work with Mac OSX.

```
> make -rpn | sed -n -e '/^$/ { n ; /^[^ ]*:/p }' | sed "s/: *.*$//g"
```

The MPAS framework supports multiple *cores* — currently a shallow water model, an ocean model, a non-hydrostatic atmosphere model, and a non-hydrostatic atmosphere initialization core — so the build process must be told which core to build. This is done by either setting the environment variable `CORE` to the name of the model core to build, or by specifying the core to be built explicitly on the command-line when running `make`. For the shallow water core, for example, one may run either

```
> setenv CORE sw
> make gfortran
```

or

```
> make gfortran CORE=sw
```

If the `CORE` environment variable is set and a core is specified on the command-line, the command-line value takes precedence; if no core is specified, either on the command line or via the `CORE` environment variable, the build process will stop with an error message stating such. Assuming compilation is successful, the model executable, named `${CORE}_model` (e.g., `sw_model`), should be created in the top-level MPAS directory.

In order to get a list of available cores, one can simply run the top-level `Makefile` without setting the `CORE` environment variable, or passing the core via the command-line. And example of the output from this can be seen below.

```
> make
```

```
( make error )
make[1]: Entering directory '/home/douglasj/Documents/svn-mpas-model.cgd.ucar.edu/trunk/mpas'

Usage: make target CORE=[core] [options]

Example targets:
ifort
gfortran
xlf
pgi

Availabe Cores:
atmosphere
init_atmosphere
landice
ocean
sw

Available Options:
DEBUG=true    - builds debug version. Default is optimized version.
USE_PAPI=true - builds version using PAPI for timers. Default is off.
TAU=true      - builds version using TAU hooks for profiling. Default is off.

Ensure that NETCDF, PNETCDF, PIO, and PAPI (if USE_PAPI=true) are environment variables
that point to the absolute paths for the libraries.

************ ERROR ************
No CORE specified. Quitting.
************ ERROR ************

make[1]: Leaving directory '/home/douglasj/Documents/svn-mpas-model.cgd.ucar.edu/trunk/mpas'
```

2.4 Cleaning

To remove all files that were created when the model was built, including the model executable itself, `make` may be run for the 'clean' target:

```
> make clean
```

As with compiling, the core to be cleaned is specified by the `CORE` environment variable, or by specifying a core explicitly on the command-line with `CORE=`.

2.5 Graph partitioning with METIS

Before MPAS can be run in parallel, a mesh decomposition file with an appropriate number of partitions (equal to the number of MPI tasks that will be used) is required in the run directory. A limited number of mesh decomposition files (`graph.info.part.*`) are provided with each test case. In order to create new mesh decomposition files for your desired number of partitions, begin with the provided `graph.info` file and partition with METIS software (http://glaros.dtc.umn.edu/gkhome/views/metis). The serial graph partitioning program, METIS (rather than ParMETIS or hMETIS) should be sufficient for quickly partitioning any SCVT produced by the grid_gen mesh generator.

After installing METIS, a `graph.info` file may be partitioned into N partitions by running

```
> gpmetis graph.info N
```

The resulting file, `graph.info.part.`N, can then be copied into the MPAS run directory before running the model with N MPI tasks.

Chapter 3

Grid Description

This chapter provides a brief introduction to the common types of grids used in the MPAS framework.

The MPAS grid system requires the definition of seven elements. These seven elements are composed of two types of *cells*, two types of *lines*, and three types of *points*. These elements are depicted in Figure 3.1 and defined in Table 3.1. These elements can be defined on either the plane or the surface of the sphere. The two types of cells form two meshes, a primal mesh composed of Voronoi regions and a dual mesh composed of Delaunay triangles. Each corner of a primal mesh cell is uniquely associated with the "center" of a dual mesh cell and vice versa. So we define the two mesh as either a primal mesh (composed of cells P_i) or a dual mesh (composed of cells D_v). The center of any primal mesh cell, P_i, is denoted by $\mathbf{x}_i$ and the center of any the dual mesh cell, D_v, is denoted by $\mathbf{x}_v$. The boundary of a given primal mesh cell P_i is composed of the set of lines that connect the $\mathbf{x}_v$ locations of associated dual mesh cells D_v. Similarly, the boundary of a given dual mesh cell D_v is composed of the set of lines that connect the $\mathbf{x}_i$ locations of the associated primal mesh cells P_i.

As shown in Figure 3.1, a line segment that connects two primal mesh cell centers is uniquely associated with a line segment that connects two dual mesh cell centers. We assume that these two line segments cross and the point of intersection is labeled as $\mathbf{x}_e$. In addition, we assume that these two line segments are orthogonal as indicated in Figure 3.1. Each $\mathbf{x}_e$ is associated with two distances: d_e measures the distance between the primal mesh cells sharing $\mathbf{x}_e$ and l_e measures the distance between the dual mesh cells sharing $\mathbf{x}_e$.

Since the two line segments crossing at $\mathbf{x}_e$ are orthogonal, these line segments form a convenient local coordinate system for each edge. At each $\mathbf{x}_e$ location a unit vector $\mathbf{n}_e$ is defined to be parallel to the line connecting primal mesh cells. A second unit vector $\mathbf{t}_e$ is defined such that $\mathbf{t}_e = \mathbf{k} \times \mathbf{n}_e$.

In addition to these seven element types, we require the definition of *sets of elements*. In all, eight different types of sets are required and these are defined and explained in Table 3.2 and Figure 3.2. The notation is always of the form of, for example, $i \in CE(e)$, where the LHS indicates the type of element to be gathered (cells) based on the RHS relation to another type of element (edges).

Table 3.3 provides the names of all *elements* and all *sets of elements* as used in the MPAS framework. Elements appear twice in the table when described in the grid file in more than one way, e.g. points are described with both cartesian and latitude/longitude coordinates. An "ncdump -h" of any MPAS grid, output or restart file will contain all variable names shown in second column of Table 3.3.

Table 3.1: Definition of elements used to build the MPAS grid.

Element	*Type*	*Definition*
$\mathbf{x}_i$	point	location of center of primal-mesh cells
$\mathbf{x}_v$	point	location of center of dual-mesh cells
$\mathbf{x}_e$	point	location of edge points where velocity is defined
d_e	line segment	distance between neighboring $\mathbf{x}_i$ locations
l_e	line segment	distance between neighboring $\mathbf{x}_v$ locations
P_i	cell	a cell on the primal-mesh
D_v	cell	a cell on the dual-mesh

Table 3.2: Definition of element groups used to reference connections in the MPAS grid. Examples are provided in Figure 3.2.

Syntax	*ouptut*
$e \in EC(i)$	set of edges that define the boundary of P_i.
$e \in EV(v)$	set of edges that define the boundary of D_v.
$i \in CE(e)$	two primal-mesh cells that share edge e.
$i \in CV(v)$	set of primal-mesh cells that form the vertices of dual mesh cell D_v.
$v \in VE(e)$	the two dual-mesh cells that share edge e.
$v \in VI(i)$	the set of dual-mesh cells that form the vertices of primal-mesh cell P_i.
$e \in ECP(e)$	edges of cell pair meeting at edge e.
$e \in EVC(v,i)$	edge pair associated with vertex v and mesh cell i.

Table 3.3: Variable names used to describe a MPAS grid.

Element	*Name*	*Size*	*Comment*
$\mathbf{x}_i$	{x,y,z}Cell	nCells	cartesian location of $\mathbf{x}_i$
$\mathbf{x}_i$	{lon,lat}Cell	nCells	longitude and latitude of $\mathbf{x}_i$
$\mathbf{x}_v$	{x,y,z}Vertex	nVertices	cartesian location of $\mathbf{x}_v$
$\mathbf{x}_v$	{lon,lat}Vertex	nVertices	longitude and latitude of $\mathbf{x}_v$
$\mathbf{x}_e$	{x,y,z}Edge	nEdges	cartesian location of $\mathbf{x}_e$
$\mathbf{x}_e$	{lon,lat}Edge	nEdges	longitude and latitude of $\mathbf{x}_e$
d_e	dcEdge	nEdges	distance between $\mathbf{x}_i$ locations
l_e	dvEdge	nEdges	distance between $\mathbf{x}_v$ locations
$e \in EC(i)$	edgesOnCell	(nEdgesMax,nCells)	edges that define P_i.
$e \in EV(v)$	edgesOnVertex	(3,nCells)	edges that define D_v.
$i \in CE(e)$	cellsOnEdge	(2,nEdges)	primal-mesh cells that share edge e.
$i \in CV(v)$	cellsOnVertex	(3,nVertices)	primal-mesh cells that define D_v.
$v \in VE(e)$	verticesOnEdge	(2,nEdges)	dual-mesh cells that share edge e.
$v \in VI(i)$	verticesOnCell	(nEdgesMax,nCells)	vertices that define P_i.

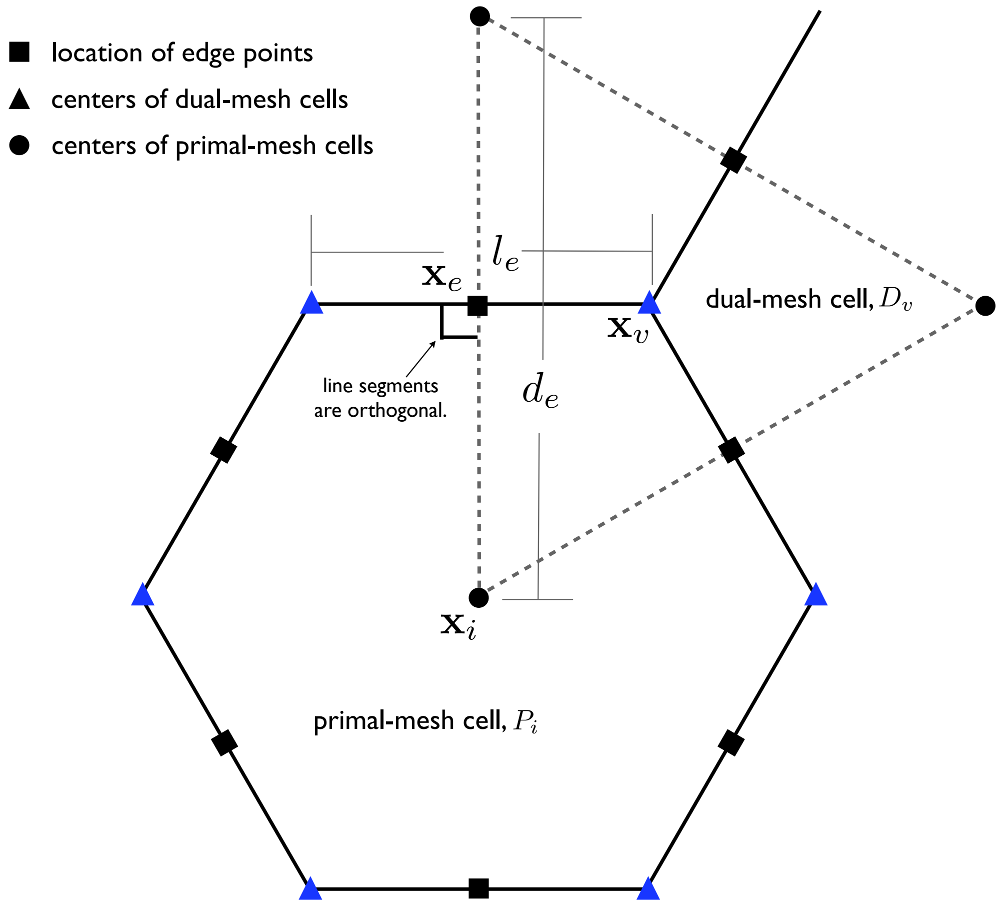
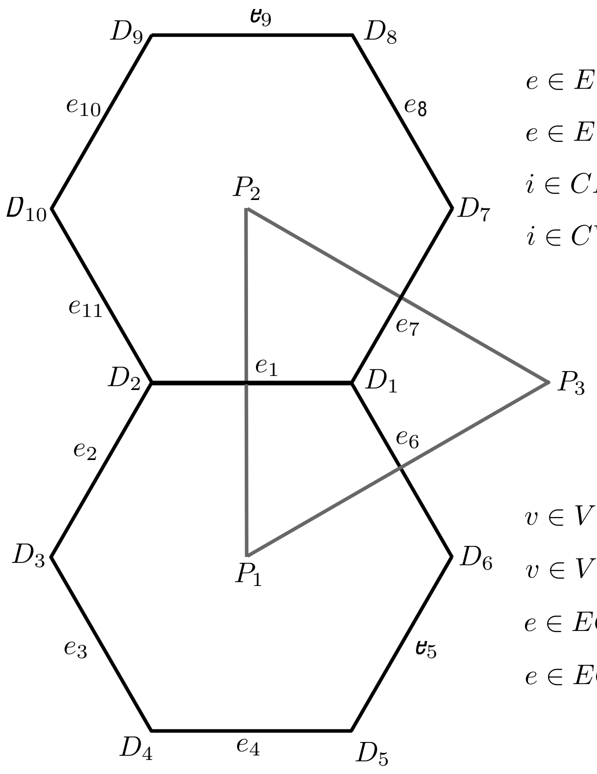

■ location of edge points
▲ centers of dual-mesh cells
● centers of primal-mesh cells
l_e
$\mathbf{x}_e$
$\mathbf{x}_v$
dual-mesh cell, D_v
line segments are orthogonal.
d_e
$\mathbf{x}_i$
primal-mesh cell, P_i

Figure 3.1: Definition of elements used to build the MPAS grid. Also see Table 3.1.



$$e \in EC(P_1) = [e_1, e_2, e_3, e_4, e_5, e_6]$$
$$e \in EV(D_1) = [e_1, e_6, e_7]$$
$$i \in CE(e_1) = [P_1, P_2]$$
$$i \in CV(D_1) = [P_1, P_2, P_3]$$

$$v \in VE(e_1) = [D_1, D_2]$$
$$v \in VC(P_1) = [D_1, D_2, D_3, D_4, D_4, D_5, D_6]$$
$$e \in ECP(e_1) = [e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}]$$
$$e \in ECV(P_1, D_1) = [e_1, e_6]$$

Figure 3.2: Definition of element groups used to reference connections in the MPAS grid. Also see Table 3.2.

Chapter 4

Visualization

This chapter discusses visualization tools that may be used by all cores. For instructions on additional visualization tools for this core, see Chapter 9.

4.1 ParaView

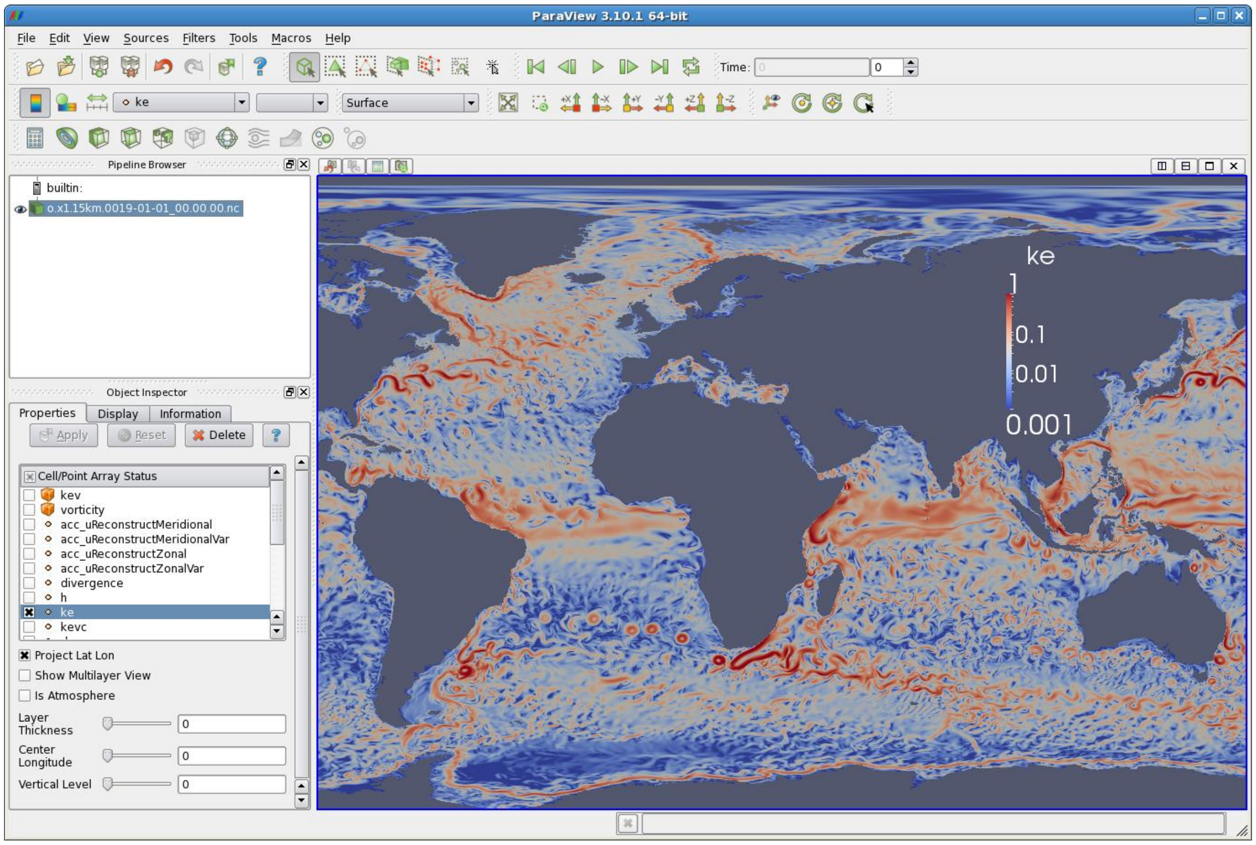
ParaView may be used to visualize MPAS initialization, output, and restart files. It includes a reader that was specifically designed to read MPAS NetCDF files, including Cartesian and spherical domains. At this time, only cell-centered quantities may be plotted with ParaView. Variables located at edges and vertices must be interpolated to cell centers for visualization.

ParaView is freely available for download at `http://www.paraview.org`. Binary installations are available for Windows, Mac, and Linux, as well as source code files and tutorials. From the ParaView website:

> ParaView is an open-source, multi-platform data analysis and visualization application. ParaView users can quickly build visualizations to analyze their data using qualitative and quantitative techniques. The data exploration can be done interactively in 3D or programmatically using ParaView's batch processing capabilities. ParaView was developed to analyze extremely large datasets using distributed memory computing resources. It can be run on supercomputers to analyze datasets of terascale as well as on laptops for smaller data.

To visualize an MPAS cell-centered variable in ParaView, open the file and choose `MPAS NetCDF (Unstructured)` as the file format. In the lower left Object Inspector panel, choose your variables of interest (Figure 4.1). For large data sets, loading fewer variables will result in less wait time. Options are available for latitude-longitude projections, vertical level, etc. Click the 'Apply' button to load the data set. In the toolbars at the top, choose the variable to plot from the pull-down menu, and 'Surface' for the type of visualization. The color bar button displays a color bar, and the color scale editor button allows the user to manually change the color bar type and extents. The Filters menu provides computational tools for interactive data manipulation. Movies, in avi format or as individual frames, may be conveniently created with the `Save Animation` tool in the File menu.

Paraview may be used to view the grid from any MPAS NetCDF file by choosing `Wireframe` or `Suface With Edges` from the visualization-type pull-down menu (Figure 4.2). This produces a view of the Delaunay triangulation, which is the dual mesh to the primal Voronoi cell grid (Figure



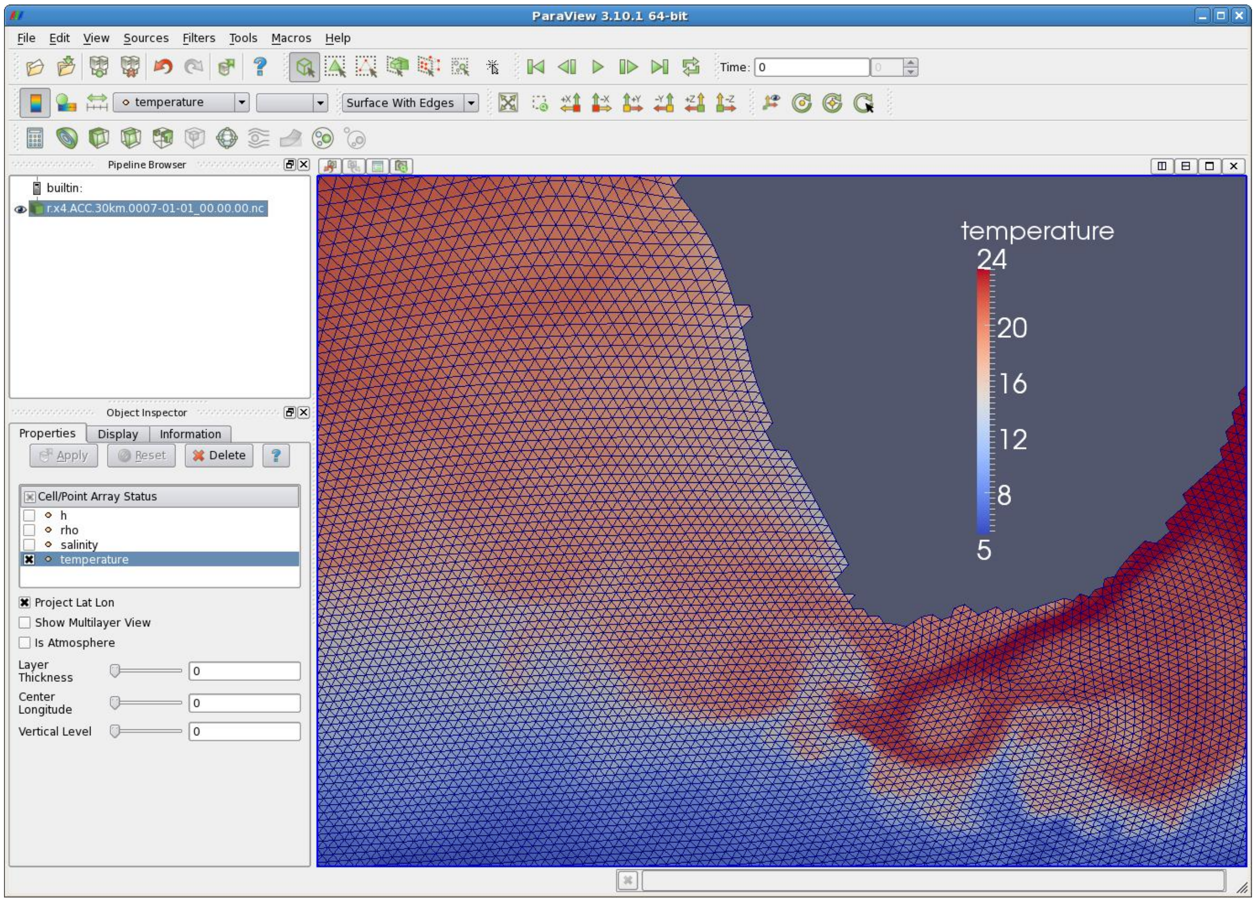
Cell/Point Array Status
- [ ] kev
- [ ] vorticity
- [ ] acc_uReconstructMeridional
- [ ] acc_uReconstructMeridionalVar
- [ ] acc_uReconstructZonal
- [ ] acc_uReconstructZonalVar
- [ ] divergence
- [ ] h
- [x] ke
- [ ] kevc

- [x] Project Lat Lon
- [ ] Show Multilayer View
- [ ] Is Atmosphere

Layer Thickness	0
Center Longitude	0
Vertical Level	0

Figure 4.1: Example of ParaView to view an MPAS NetCDF file.

3.1). Paraview plots all variables by interpolating colors between each corner of the Delaunay triangles. These corners are the cell-center locations of the primal grid.



Time: 0

temperature

Surface With Edges

builtin:

r.x4.ACC.30km.0007-01-01_00.00.00.nc

Cell/Point Array Status

- [ ] h
- [ ] rho
- [ ] salinity
- [x] temperature

- [x] Project Lat Lon
- [ ] Show Multilayer View
- [ ] Is Atmosphere

Layer Thickness 0

Center Longitude 0

Vertical Level 0

Figure 4.2: Example of visualizing the dual mesh from an MPAS NetCDF file.

Part II

MPAS-Land Ice

Chapter 5

Governing Equations

Advection is performed on a C-grid, with scalar quantities (thickness, temperature, age, etc.) on the Voronoi cell centers and velocities and fluxes centered at Voronoi cell edges. MPAS-Land Ice uses SI units everywhere, including input and output. One exception is the ability (but not the requirement) to specify the model time step in years (but which is then converted to seconds internally).

5.1 Momentum Balance

Currently within MPAS-Land Ice, the momentum balance for ice is approximated with the Shallow Ice Approximation (SIA) (Hutter, 1983), which is solved explicitly. In terms of balancing the gravitational body force, the SIA neglects all but the 0th-order, vertical shear-stress gradients. The preferred numerical approach for implementing the SIA in ice sheet models is not to solve for the velocity directly but to instead formulate a parabolic PDE describing the thickness evolution, with velocities implicit in the formulation. However, for higher-order treatments of the momentum balance, it is necessary to solve the velocity and thickness evolution steps separately. Therefore, to allow for the eventual incorporation of higher-order velocity solvers in MPAS Land Ice, the current design explicitly calculates velocities from the SIA.

Within a column, at any point in the model domain in map view, the depth-dependent SIA velocity can be solved for as:

$$\mathbf{u}(z) = -\frac{1}{2}A(\rho g)^3(\nabla s)^3\left[H^4 - (h-z)^4\right] \tag{5.1}$$

where $\mathbf{u}(z)$ is the horizontal velocity vector, A is the flow rate factor (primarily a function of ice temperature), ρ is the density of ice, g is acceleration due to gravity, s is the ice surface elevation, H is ice thickness, and z is the vertical coordinate. Velocities are (nonlinearly) proportional to both the ice thickness and the ice surface slope.

Velocities and fluxes are calculated on the midpoint of Voronoi cell edges. Surface slope is calculated on cell edges based on surface elevation at adjacent cell centers. Ice thickness on edges is calculated as the average of the adjacent cell center values (2nd-order approximation).

5.2 Time Integration

Currently, MPAS Land Ice only supports Forward Euler time integration.

5.3 Advection

Currently, MPAS Land Ice only supports advection of thickness and only using First-Order Upwinding. In 1D, first-order upwinding of ice thickness using a Forward Euler time step is described by:

$$\frac{H_i^{n+1} - H_i^n}{\Delta t} + u\frac{H_i^n - H_{i-1}^n}{\Delta x} = 0 \quad \text{for} \quad u > 0 \tag{5.2}$$

$$\frac{H_i^{n+1} - H_i^n}{\Delta t} + u\frac{H_{i+1}^n - H_i^n}{\Delta x} = 0 \quad \text{for} \quad u < 0 \tag{5.3}$$

where Δx represents the horizontal grid spacing along flow, subscripts designate the spatial dimension, and superscripts designate the time dimension.

(See, e.g. http://en.wikipedia.org/wiki/Upwind_scheme)

To allow the eventual inclusion of tracer advection, thickness is advected level-by-level, rather than the cheaper operation of advecting the total column thickness.

Chapter 6

Dimensions

Name	**Units**	**Description**
nCells	*unitless*	The number of polygons in the primary grid.
nEdges	*unitless*	The number of edge midpoints in either the primary or dual grid.
maxEdges	*unitless*	The largest number of edges any polygon within the grid has.
maxEdges2	*unitless*	Two times the largest number of edges any polygon within the grid has.
nVertices	*unitless*	The total number of cells in the dual grid. Also the number of corners in the primary grid.
TWO	*unitless*	The number two as a dimension.
R3	*unitless*	The number three as a dimension.
vertexDegree	*unitless*	The number of cells or edges touching each vertex.
nVertLevels	*unitless*	The number of levels in the vertical direction. All vertical levels share the same horizontal locations.
nVertLevelsP1	*unitless*	The number of interfaces in the vertical direction.

Chapter 7

Namelist options

Embedded links point to more detailed namelist information in the appendix.

7.1 velocity_solver

The velocity_solver namelist record controls which velocity solver is used and options associated with velocity solvers.

Name	Description
config_velocity_solver	Selection of the method for solving ice velocity.

7.2 advection

The advection namelist record controls options assocated with advection of thickness and tracers. Tracer advection is not currently supported.

Name	Description
config_thickness_advection	Selection of the method for advecting thickness.
config_tracer_advection	Selection of the method for advecting tracers.

7.3 physical_parameters

The physical_parameters namelist record sets scalar physical parameters and constants within the land ice model.

Name	Description
config_ice_density	ice density to use
config_ocean_density	ocean density to use for calculating floatation
config_sea_level	sea level to use for calculating floatation

config_default_flowParamA	Defines the default value of the flow law parameter A to be used if it is not being calculated from ice temperature. Defaults to the SI representation of 1.0e-16 yr^{-1} Pa^{-3}.
config_flowLawExponent	Defines the value of the Glen flow law exponent, n.
config_dynamic_thickness	Defines the ice thickness below which dynamics are not calculated.

7.4 time_integration

The time integration namelist record controls parameters that pertain to all time-stepping methods. At present, Forward Euler is the only time integration method implemented.

Name	Description
config_dt_years	Length of model time-step in years. Will be used instead of config_dt_seconds if greater than zero. Currently the model assumes there are 365.0 * 24.0 * 3600.0 seconds in a year and the calendar type is not considered for this conversion.
config_dt_seconds	Length of model time-step in seconds. This value will only be used if config_dt_years is less than or equal to zero.
config_time_integration	Time integration method.

7.5 time_management

General time management is handled by the time_management namelist record. Included options handle time-related parts of MPAS, such as the calendar type and if the simulation is a restart or not.

Users should use this record to specify the beginning time of the simulation, and either the duration or the end of the simulation. Only the end or the duration need to be specified as the other is derived within MPAS from the beginning time and other specified one.

If both the run duration and stop time are specified, run duration is used in place of stop time.

Name	Description
config_do_restart	Determines if the initial conditions should be read from a restart file, or an input file. To perform a restart, simply set this to true in the namelist.input file and modify the start time to be the time you want restart from. A restart will read the grid information from the input field, and the restart state from the restart file. It will perform a run normally, i.e. do all the same init.
config_start_time	Timestamp describing the initial time of the simulation. If it is set to 'file', the initial time is read from restart_timestamp
config_stop_time	Timestamp describing the final time of the simulation. If it is set to 'none' the final time is determined from config_start_time and config_run_duration. If config_run_duration is also specified, it takes precedence over config_stop_time. Set config_stop_time to be equal to config_start_time (and config_run_duration to 'none') to perform a diagnostic solve of velocity.

config_run_duration	Timestamp describing the length of the simulation. If it is set to 'none' the duration is determined from config_start_time and config_stop_time. config_run_duration overrides inconsistent values of config_stop_time. If a time value is specified for config_run_duration, it must be greater than 0.
config_calendar_type	Selection of the type of calendar that should be used in the simulation.

7.6 io

The io namelist record provides options for modifications to the I/O system of MPAS. These include frequency, file name, and parallelization options.

Name	**Description**
config_input_name	The path to the input file for the simulation.
config_output_name	The template path and name to the output file from the simulation. A time stamp is prepended to the extension of the file (.nc).
config_restart_name	The template path and name to the restart file for the simulation. A time stamp is prepended to the extension of the file (.nc) both for input and output.
config_restart_timestamp_name	The name of the file to which the timestamp of the latest restart file is written. This file is subsequently used to set the start time when config_start_time is set to 'file' and config_do_restart is set to .true.
config_restart_interval	Timestamp determining how often a restart file should be written. Currently years and months are not supported, so you have to specify the restart interval in units of days! ** We could eventually propose a change to framework to fix this in subroutine mpas_set_timeInterval in mpas_timekeeping module.
config_output_interval	Timestamp determining how often an output file should be written.
config_stats_interval	Timestamp determining how often a global statistics files should be written.
config_write_stats_on_startup	Logical flag determining if statistics files should be written prior to the first time step.
config_write_output_on_startup	Logical flag determining if an output file should be written prior to the first time step.
config_frames_per_outfile	Integer specifying how many time frames should be included in an output file. Once the maximum is reached, a new output file is created. If 0 (or less) is specified then all time frames are included in a single file called 'output.nc'.
config_pio_num_iotasks	Integer specifying how many IO tasks should be used within the PIO library. A value of 0 causes all MPI tasks to also be IO tasks. IO tasks are required to write contiguous blocks of data to a file.
config_pio_stride	Integer specifying the stride of each IO task.

7.7 decomposition

MPAS handles decomposing all variables into computational blocks. The decomposition used needs to be specified at run time and is computed by an external tool (e.g. metis). Additionally, MPAS supports multiple computational blocks per MPI process, and the user may specify an additional decomposition file which can specify the assignment of blocks to MPI processes. Run-time parameters that control the run-time decomposition used are specified within the decomposition namelist record.

Name	**Description**
config_num_halos	Determines the number of halo cells extending from a blocks owned cells (Called the 0-Halo). The default of 3 is the minimum that can be used with monotonic advection.
config_block_decomp_file_prefix	Defines the prefix for the block decomposition file. Can include a path. The number of blocks is appended to the end of the prefix at run-time.
config_number_of_blocks	Determines the number of blocks a simulation should be run with. If it is set to 0, the number of blocks is the same as the number of MPI tasks at run-time.
config_explicit_proc_decomp	Determines if an explicit processor decomposition should be used. This is only useful if multiple blocks per processor are used.
config_proc_decomp_file_prefix	Defines the prefix for the processor decomposition file. This file is only read if config_explicit_proc_decomp is .true. The number of processors is appended to the end of the prefix at run-time.

7.8 debug

At run-time a user can enable debugging features within MPAS-Land Ice. Currently the only debug option is to print more detailed information about thickness advection. Potential future debug options would be to include disabling of any tendencies to help determine why an issue might be happening; various checks on certain fields; and the ability to prescribe both a thickness and velocity field at run-time which are constant throughout a simulation. All options that control these debugging features are specified within the debug namelist record.

Name	**Description**
config_print_thickness_advection_info	Prints additional information about thickness advection.

Chapter 8

Variable definitions

Embedded links point to more detailed variable information in the appendix.

8.1 state

The state data structure contains a set of prognostic and diagnostic fields that are time dependent. The fields contained inside of state have two time levels available. (More than two is possible within the MPAS Framework, but only two have been implemented in the Land Ice core.)

Name	**Description**
xtime	model time, with format 'YYYY-MM-DD_HH:MM:SS'
thickness	ice thickness
layerThickness	layer thickness
temperature	ice temperature
lowerSurface	elevation at bottom of ice
upperSurface	elevation at top of ice
layerThicknessEdge	layer thickness on cell edges
cellMask	bitmask indicating various properties about the ice sheet on cells. cellMask only needs to be a restart field if config_allow_additional_advance = false (to keep the mask of initial ice extent)
edgeMask	bitmask indicating various properties about the ice sheet on edges.
vertexMask	bitmask indicating various properties about the ice sheet on vertices.
normalVelocity	horizonal velocity, normal component to an edge
uReconstructX	x-component of velocity reconstructed on cell centers
uReconstructY	y-component of velocity reconstructed on cell centers
uReconstructZ	z-component of velocity reconstructed on cell centers
uReconstructZonal	zonal velocity reconstructed on cell centers
uReconstructMeridional	meridional velocity reconstructed on cell centers

8.2 tend

The tend data structure represents the tendencies used to time step the prognostic variables within the state structure.

Name	Description
tend_layerThickness	time tendency of layer thickness
tend_temperature	time tendency of ice temperature

8.3 mesh

The mesh data type contains a single time level. The fields inside the mesh structure are not assumed to be time dependent. This data structure contains fields that describe the mesh, and the connectivity of the mesh. Most of the fields contained in this structure are shared throughout all MPAS cores. Additionally, a few Land Ice specific variables (that are time-independent) are stored here, but may be moved in the future.

Name	Description
latCell	Latitude location of cell centers in radians.
lonCell	Longitude location of cell centers in radians.
xCell	X Coordinate in cartesian space of cell centers.
yCell	Y Coordinate in cartesian space of cell centers.
zCell	Z Coordinate in cartesian space of cell centers.
indexToCellID	List of global cell IDs.
latEdge	Latitude location of edge midpoints in radians.
lonEdge	Longitude location of edge midpoints in radians.
xEdge	X Coordinate in cartesian space of edge midpoints.
yEdge	Y Coordinate in cartesian space of edge midpoints.
zEdge	Z Coordinate in cartesian space of edge midpoints.
indexToEdgeID	List of global edge IDs.
latVertex	Latitude location of vertices in radians.
lonVertex	Longitude location of vertices in radians.
xVertex	X Coordinate in cartesian space of vertices.
yVertex	Y Coordinate in cartesian space of vertices.
zVertex	Z Coordinate in cartesian space of vertices.
indexToVertexID	List of global vertex IDs.
cellsOnEdge	List of cells that straddle each edge.
nEdgesOnCell	Number of edges that border each cell.
nEdgesOnEdge	Number of edges that surround each of the cells that straddle each edge. These edges are used to reconstruct the tangential velocities.
edgesOnCell	List of edges that border each cell.
edgesOnEdge	List of edges that border each of the cells that straddle each edge.
weightsOnEdge	Reconstruction weights associated with each of the edgesOnEdge.
dvEdge	Length of each edge, computed as the distance between verticesOnEdge.
dcEdge	Length of each edge, computed as the distance between cellsOnEdge.
angleEdge	Angle the edge normal makes with local eastward direction.
areaCell	Area of each cell in the primary grid.
areaTriangle	Area of each cell (triangle) in the dual grid.
edgeNormalVectors	Normal vector defined at an edge.
localVerticalUnitVectors	Unit surface normal vectors defined at cell centers.
cellTangentPlane	The two vectors that define a tangent plane at a cell center.

cellsOnCell	List of cells that neighbor each cell.
verticesOnCell	List of vertices that border each cell.
verticesOnEdge	List of vertices that straddle each edge.
edgesOnVertex	List of edges that share a vertex as an endpoint.
cellsOnVertex	List of cells that share a vertex.
kiteAreasOnVertex	Area of the portions of each dual cell that are part of each cellsOnVertex.
coeffs_reconstruct	Coefficients to reconstruct velocity vectors at cells centers.
edgeSignOnCell	Sign of edge contributions to a cell for each edge on cell. Used for bit-reproducible loops. Represents directionality of vector connecting cells.
edgeSignOnVertex	Sign of edge contributions to a vertex for each edge on vertex. Used for bit-reproducible loops. Represents directionality of vector connecting vertices.
layerThicknessFractions	Fractional thickness of each sigma layer
layerCenterSigma	Sigma (fractional) level at center of each layer
layerInterfaceSigma	Sigma (fractional) level at interface between each layer (including top and bottom)
bedTopography	Elevation of ice sheet bed. Once isostasy is added to the model, this should become a state variable.
sfcMassBal	Surface mass balance

Chapter 9

Land Ice Visualization

This chapter discusses visualization tools that are specific to the Land Ice core. For instructions on visualization tools that may be used by all cores, such as Paraview, see Chapter 4.

9.1 Python

Python visualization scripts are available for the dome test case, and general python visualization tools are in development. In order to use these scripts, the following python modules are required:

- matplotlib, see `http://matplotlib.org`
- numpy, see `http://www.numpy.org`
- pylab, see `www.scipy.org`
- netCDF4, see `http://code.google.com/p/netcdf4-python`

Most package managers (including MacPorts) have packages for these python modules. Another convenient way to install all these libraries at once is to purchase the Enthought Python Distribution (EPD), available at `https://www.enthought.com/products/epd`. Many institutions have Python-EPD installed on their compute clusters.

Chapter 10

Test Cases

Eventually test cases will be available for download. Currently they are only part of the Development code for MPAS-Land Ice.

10.1 Halfar Dome

This test case describes the time evolution of a dome of ice as described by Halfar (1983). This test provide an analytic solution for a flat-bedded SIA problem.

$$\frac{\partial H}{\partial t} = \nabla \cdot (\Gamma H^{n+2} |\nabla H|^{n-1} \nabla H) \tag{10.1}$$

where n is the exponent in the Glen flow law, commonly taken as 3, and Γ is a positive constant:

$$\Gamma = \frac{2}{n+2} A(\rho g)^n \tag{10.2}$$

For $n = 3$, this reduces to:

$$H(t,r) = H_0 \left(\frac{t_0}{t}\right)^{\frac{1}{9}} \left[1 - \left(\left(\frac{t_0}{t}\right)^{\frac{1}{18}} \frac{r}{R_0}\right)^{\frac{4}{3}}\right]^{\frac{3}{7}} \tag{10.3}$$

where

$$t_0 = \frac{1}{18\Gamma} \left(\frac{7}{4}\right)^3 \frac{R_0^4}{H_0^7} \tag{10.4}$$

and H_0, R_0 are the central height of the dome and its radius at time $t = t_0$.

For more details see http://www.projects.science.uu.nl/iceclimate/karthaus/2009/more/lecturenotes/EdBueler.pdf, Bueler et al. (2005), Halfar (1983).

10.1.1 Provided Files

Our implementation of the Halfar dome has an initial radius of $R_0 = 21.2$ km and an initial thickness of $H = 707.1$ m. These values can be changed by editing `setup_dome_initial_conditions.py`.

- readme.txt:
  Information about the test case.

- namelist.input.periodic_hex:
  This is the namelist file to use for creating the grid file for the run.
  It is used with the `periodic_hex` grid generator.
  It should be renamed to `namelist.config` when executing `periodic_hex`.
  periodic_hex will be used to generate `grid.nc` which can be used to create
  `landice_grid.nc` and `graph.info.part.*` which can be used for running the model
  on more than one processor.

- setup_dome_initial_conditions.py:
  This python script generates the dome initial condition after
  an empty `landice_grid.nc` file exists.

- namelist.input.landice_core:
  This is the namelist file to use for running the model with `landice_model`.
  It should be renamed to `namelist.config` when executing `landice_model`.

- halfar.py:
  This is the script to compare model results to the analytic solution.

- visualize_dome.py:
  This python script provides some general visualization of the model output. It can be used in addition to `halfar.py` for additional visualization.

10.1.2 Results

As the dome of ice evolves, its margin advances and its thickness decreases (there is no surface mass balance to add new mass). The script `halfar.py` will plot the modeled and analytic thickness at a specified time (Figure 10.1), as well as report model error statistics. Invoke `halfar.py --help` for details of its usage.

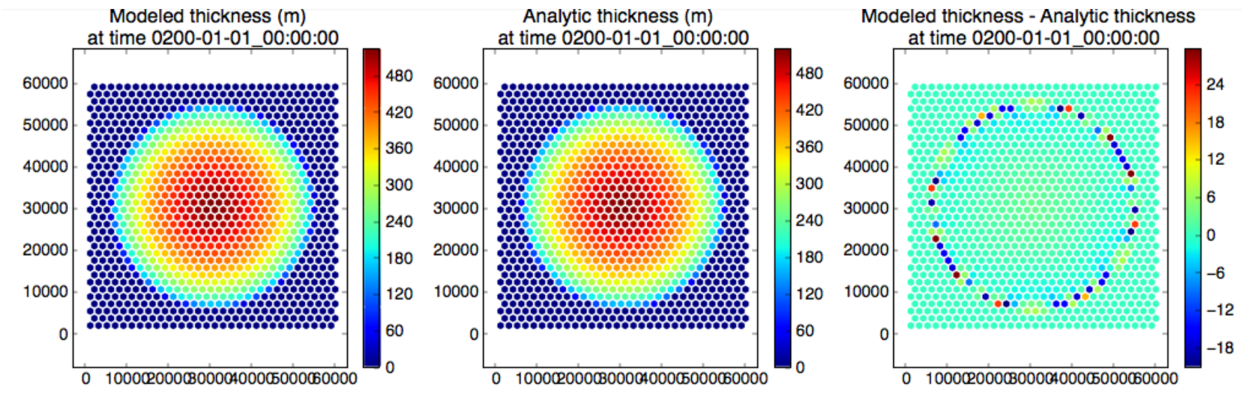


Figure 10.1: Halfar test case results after 200 years of dome evolution. This figure is generated by `halfar.py`.

10.2 EISMINT-1 Test Cases

This test case is from the European Ice Sheet Modelling INiTiative intercomparison experiments. These experiments are described at `http://homepages.vub.ac.be/~phuybrec/eismint.html` and in Huybrechts et al. (1996).

Currently only the Moving Margin 1 Test Case from EISMINT-1 is included.

10.2.1 Provided Files

- namelist.input.periodic_hex
  This file is used for running periodic_hex to create a grid for the test case. It needs to be renamed to 'namelist.input' to run periodic_hex) if mesh needs to be generated. If you downloaded a tar archive of this test case, you do not need to create the mesh and can ignore this file.

- namelist.input / namelist.input.landice_core
  This file is used for actually running dome test case in the MPAS land ice core. It includes most but not all options available to the model. See the default namelist.input.landice file in the MPAS root directory for a list of all options available. They are also documented in the User's Guide. If you downloaded a tar archive of this test case, this should be called namelist.input.

- setup_initial_conditions_EISMINT1-MovingMargin-1.py
  This file can be used to setup the initial conditions for the test case. If you downloaded a tar archive, you do not need to do this. However, if you want to modify the IC for some reason, you can edit and run this script.

- check_output_eismint-mm1.py
  This script can be used to compare model output to results from the EISMINT intercomparison.

10.2.2 Results

As the initial ice sheet evolves, its shape eventually reaches a steady-state with the imposed surface mass balance. The script `check_output_eismint-mm1.py` will plot the modeled thickness at a specified time, as well as compare the model results to the results from the original EISMINT intercomparison. Invoke `check_output_eismint-mm1.py --help` for details of its usage.

10.3 Real World Test Cases

Eventually grids for real-world Greenland and Antarctica will be provided at varying resolutions.

Chapter 11

Global Statistics

Eventually global statistics will be calculated within MPAS Land Ice.

Chapter 12

Running MPAS-Land Ice within a coupled climate model

Eventually MPAS-Land Ice will be coupled within global climate models.

Chapter 13

Troubleshooting

13.1 Choice of time step

Symptoms: "Error in calculating thickness tendency (possibly CFL violation)" appears in log.0000.err file.

Possible cause: Time step is too long.

Remedy: Shorten time step.

Discussion: The time step must be short enough that the CFL criterion is not violated. Eventually an adaptive time integrator will be added to MPAS-Land Ice.

Chapter 14

Known Issues

- Plotting of periodic field with Paraview
- Paraview will not recognize fields without a vertical dimension (e.g. thickness will not be recognized). Current nightly builds of Paraview have fixed this problem and this functionality should be available in the next release.

Part III

Bibliography

Bibliography

Bueler, E., C. S. Lingle, J. a. Kallen-Brown, D. N. Covey, and L. N. Bowman, 2005: Exact solutions and verification of numerical models for isothermal ice sheets. *Journal of Glaciology*, **51**, 291–306, doi:10.3189/172756505781829449.
URL http://openurl.ingenta.com/content/xref?genre=article&issn=0022-1430&volume=51&issue=173&spage=291

Edwards, T. L., X. Fettweis, O. Gagliardini, F. Gillet-Chaulet, H. Goelzer, J. M. Gregory, M. Hoffman, P. Huybrechts, A. J. Payne, M. Perego, S. Price, A. Quiquet, and C. Ritz., 2013: Effect of uncertainty in surface mass balance elevation feedback on projections of the future sea level contribution of the Greenland ice sheet - Part 2: Projections. *The Cryosphere Discussions*, **7**, 675–708.

Halfar, P., 1983: On the Dynamics of the Ice Sheets 2. *Journal of Geophysical Research*, **88**, 6043–6051.

Hutter, K., 1983: *Theoretical glaciology; material science of ice and the mechanics of glaciers and ice sheets*. Reidel Publishing Co., Terra Scientific Publishing Co., Tokyo.

Huybrechts, P., T. Payne, and T. E. I. Group, 1996: The EISMINT benchmarks for testing ice-sheet models. *Annals of Glaciology*, **23**, 1–12.

Leng, W., L. Ju, M. Gunzburger, S. Price, and T. Ringler, 2012: A parallel high-order accurate finite element nonlinear Stokes ice sheet model and benchmark experiments. *Journal of Geophysical Research*, **117**, F01001, doi:10.1029/2011JF001962.

Perego, M., M. Gunzburger, and J. Burkardt, 2012: Parallel finite-element implementation for higher-order ice-sheet models. *Journal of Glaciology*, **58**, 76–88, doi:10.3189/2012JoG11J063.
URL http://www.igsoc.org/journal/current/207/t11J063.pdf

Shannon, S. R., A. J. Payne, I. D. Bartholomew, M. R. V. D. Broeke, T. L. Edwards, A. J. Sole, R. S. W. V. D. Wal, and T. Zwinger, 2013: Enhanced basal lubrication and the contribution of the Greenland ice sheet to future sea-level rise. *Proceedings of the National Academy of Sciences of the United States of America*, 1–6, doi:10.1073/pnas.1212647110.

Part IV

Appendices

Appendix A

Namelist options

Embedded links point to information in chapter 7

A.1 velocity_solver

A.1.1 config_velocity_solver

Type:	character
Units:	*unitless*
Default Value:	sia
Possible Values:	'sia'

Table A.1: config_velocity_solver: Selection of the method for solving ice velocity.

A.2 advection

A.2.1 config_thickness_advection

Type:	character
Units:	*unitless*
Default Value:	fo
Possible Values:	'fo', 'none'

Table A.2: config_thickness_advection: Selection of the method for advecting thickness.

A.2.2 config_tracer_advection

Type:	character
Units:	*unitless*
Default Value:	none

Possible Values:	'none'

Table A.3: config_tracer_advection: Selection of the method for advecting tracers.

A.3 physical_parameters

A.3.1 config_ice_density

Type:	real
Units:	$kg\ m^{-3}$
Default Value:	910.0
Possible Values:	Any positive real value

Table A.4: config_ice_density: ice density to use

A.3.2 config_ocean_density

Type:	real
Units:	$kg\ m^{-3}$
Default Value:	1028.0
Possible Values:	Any positive real value

Table A.5: config_ocean_density: ocean density to use for calculating floatation

A.3.3 config_sea_level

Type:	real
Units:	$m\ above\ datum$
Default Value:	0.0
Possible Values:	Any real value

Table A.6: config_sea_level: sea level to use for calculating floatation

A.3.4 config_default_flowParamA

Type:	real
Units:	$s^{-1}\ Pa^{-n}$

Default Value:	3.1709792e-24
Possible Values:	Any positive real value

Table A.7: config_default_flowParamA: Defines the default value of the flow law parameter A to be used if it is not being calculated from ice temperature. Defaults to the SI representation of 1.0e-16 yr^{-1} Pa^{-3}.

A.3.5 config_flowLawExponent

Type:	real
Units:	*none*
Default Value:	3.0
Possible Values:	Any real value

Table A.8: config_flowLawExponent: Defines the value of the Glen flow law exponent, n.

A.3.6 config_dynamic_thickness

Type:	real
Units:	*m of ice*
Default Value:	100.0
Possible Values:	Any positive real value

Table A.9: config_dynamic_thickness: Defines the ice thickness below which dynamics are not calculated.

A.4 time_integration

A.4.1 config_dt_years

Type:	real
Units:	*yr*
Default Value:	0.5
Possible Values:	Any positive real value, but limited by CFL condition.

Table A.10: config_dt_years: Length of model time-step in years. Will be used instead of config_dt_seconds if greater than zero. Currently the model assumes there are 365.0 * 24.0 * 3600.0 seconds in a year and the calendar type is not considered for this conversion.

A.4.2 config_dt_seconds

Type:	real
Units:	s
Default Value:	0.0
Possible Values:	Any positive real value, but limited by CFL condition.

Table A.11: config_dt_seconds: Length of model time-step in seconds. This value will only be used if config_dt_years is less than or equal to zero.

A.4.3 config_time_integration

Type:	character
Units:	*unitless*
Default Value:	forward_euler
Possible Values:	'forward_euler'

Table A.12: config_time_integration: Time integration method.

A.5 time_management

A.5.1 config_do_restart

Type:	logical
Units:	*unitless*
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.13: config_do_restart: Determines if the initial conditions should be read from a restart file, or an input file. To perform a restart, simply set this to true in the namelist.input file and modify the start time to be the time you want restart from. A restart will read the grid information from the input field, and the restart state from the restart file. It will perform a run normally, i.e. do all the same init.

A.5.2 config_start_time

Type:	character

Units:	*unitless*
Default Value:	0000-01-01_00:00:00
Possible Values:	'YYYY-MM-DD_HH:MM:SS'

Table A.14: config_start_time: Timestamp describing the initial time of the simulation. If it is set to 'file', the initial time is read from restart_timestamp

A.5.3 config_stop_time

Type:	character
Units:	*unitless*
Default Value:	0000-01-01_00:00:00
Possible Values:	'YYYY-MM-DD_HH:MM:SS' or 'none'

Table A.15: config_stop_time: Timestamp describing the final time of the simulation. If it is set to 'none' the final time is determined from config_start_time and config_run_duration. If config_run_duration is also specified, it takes precedence over config_stop_time. Set config_stop_time to be equal to config_start_time (and config_run_duration to 'none') to perform a diagnostic solve of velocity.

A.5.4 config_run_duration

Type:	character
Units:	*unitless*
Default Value:	none
Possible Values:	'DDDD_HH:MM:SS' or 'none'

Table A.16: config_run_duration: Timestamp describing the length of the simulation. If it is set to 'none' the duration is determined from config_start_time and config_stop_time. config_run_duration overrides inconsistent values of config_stop_time. If a time value is specified for config_run_duration, it must be greater than 0.

A.5.5 config_calendar_type

Type:	character
Units:	*unitless*
Default Value:	gregorian_noleap
Possible Values:	'gregorian', 'gregorian_noleap', or '360day'

Table A.17: config_calendar_type: Selection of the type of calendar that should be used in the simulation.

A.6 io

A.6.1 config_input_name

Type:	character
Units:	*unitless*
Default Value:	landice_grid.nc
Possible Values:	path/to/grid.nc

Table A.18: config_input_name: The path to the input file for the simulation.

A.6.2 config_output_name

Type:	character
Units:	*unitless*
Default Value:	output.nc
Possible Values:	path/to/output.nc

Table A.19: config_output_name: The template path and name to the output file from the simulation. A time stamp is prepended to the extension of the file (.nc).

A.6.3 config_restart_name

Type:	character
Units:	*unitless*
Default Value:	restart.nc
Possible Values:	path/to/restart.nc

Table A.20: config_restart_name: The template path and name to the restart file for the simulation. A time stamp is prepended to the extension of the file (.nc) both for input and output.

A.6.4 config_restart_timestamp_name

Type:	character
Units:	*unitless*
Default Value:	restart_timestamp
Possible Values:	path/to/restart_timestamp

Table A.21: config_restart_timestamp_name: The name of the file to which the timestamp of the latest restart file is written. This file is subsequently used to set the start time when config_start_time is set to 'file' and config_do_restart is set to .true.

A.6.5 config_restart_interval

Type:	character
Units:	*unitless*
Default Value:	3650_00:00:00
Possible Values:	'DDDD_HH:MM:SS'

Table A.22: config_restart_interval: Timestamp determining how often a restart file should be written. Currently years and months are not supported, so you have to specify the restart interval in units of days! ** We could eventually propose a change to framework to fix this in subroutine mpas_set_timeInterval in mpas_timekeeping module.

A.6.6 config_output_interval

Type:	character
Units:	*unitless*
Default Value:	0001_00:00:00
Possible Values:	'DDDD_HH:MM:SS'

Table A.23: config_output_interval: Timestamp determining how often an output file should be written.

A.6.7 config_stats_interval

Type:	character
Units:	*unitless*
Default Value:	0000_01:00:00
Possible Values:	'DDDD_HH:MM:SS'

Table A.24: config_stats_interval: Timestamp determining how often a global statistics files should be written.

A.6.8 config_write_stats_on_startup

Type:	logical
Units:	*unitless*
Default Value:	.true.
Possible Values:	.true. or .false.

Table A.25: config_write_stats_on_startup: Logical flag determining if statistics files should be written prior to the first time step.

A.6.9 config_write_output_on_startup

Type:	logical
Units:	*unitless*
Default Value:	.true.
Possible Values:	.true. or .false.

Table A.26: config_write_output_on_startup: Logical flag determining if an output file should be written prior to the first time step.

A.6.10 config_frames_per_outfile

Type:	integer
Units:	*unitless*
Default Value:	0
Possible Values:	Any integer value

Table A.27: config_frames_per_outfile: Integer specifying how many time frames should be included in an output file. Once the maximum is reached, a new output file is created. If 0 (or less) is specified then all time frames are included in a single file called 'output.nc'.

A.6.11 config_pio_num_iotasks

Type:	integer
Units:	*unitless*
Default Value:	0
Possible Values:	Any positive integer value greater than or equal to 0.

Table A.28: config_pio_num_iotasks: Integer specifying how many IO tasks should be used within the PIO library. A value of 0 causes all MPI tasks to also be IO tasks. IO tasks are required to write contiguous blocks of data to a file.

A.6.12 config_pio_stride

Type:	integer
Units:	*unitless*
Default Value:	1
Possible Values:	Any positive integer value greater than 0.

Table A.29: config_pio_stride: Integer specifying the stride of each IO task.

A.7 decomposition

A.7.1 config_num_halos

Type:	integer
Units:	*unitless*
Default Value:	3
Possible Values:	Any positive interger value.

Table A.30: config_num_halos: Determines the number of halo cells extending from a blocks owned cells (Called the 0-Halo). The default of 3 is the minimum that can be used with monotonic advection.

A.7.2 config_block_decomp_file_prefix

Type:	character
Units:	*unitless*
Default Value:	graph.info.part.
Possible Values:	Any path/prefix to a block decomposition file.

Table A.31: config_block_decomp_file_prefix: Defines the prefix for the block decomposition file. Can include a path. The number of blocks is appended to the end of the prefix at run-time.

A.7.3 config_number_of_blocks

Type:	integer
Units:	*unitless*
Default Value:	0
Possible Values:	Any integer >= 0.

Table A.32: config_number_of_blocks: Determines the number of blocks a simulation should be run with. If it is set to 0, the number of blocks is the same as the number of MPI tasks at run-time.

A.7.4 config_explicit_proc_decomp

Type:	logical
Units:	*unitless*
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.33: config_explicit_proc_decomp: Determines if an explicit processor decomposition should be used. This is only useful if multiple blocks per processor are used.

A.7.5 config_proc_decomp_file_prefix

Type:	character
Units:	*unitless*
Default Value:	graph.info.part.
Possible Values:	Any path/prefix to a processor decomposition file.

Table A.34: config_proc_decomp_file_prefix: Defines the prefix for the processor decomposition file. This file is only read if config_explicit_proc_decomp is .true. The number of processors is appended to the end of the prefix at run-time.

A.8 debug

A.8.1 config_print_thickness_advection_info

Type:	logical

Units:	*unitless*
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.35: config_print_thickness_advection_info: Prints additional information about thickness advection.

Appendix B

Variable definitions

Embedded links point to information in chapter 8

B.1 state

B.1.1 xtime

Type:	text
Units:	*unitless*
Dimension:	Time
Persistence:	persistent
Default Streams:	Restart Output
Location in code:	domain % blocklist % state % time_levs(:) % state % xtime

Table B.1: xtime: model time, with format 'YYYY-MM-DD_HH:MM:SS'

B.1.2 thickness

Type:	real
Units:	m
Dimension:	nCells Time
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % state % time_levs(:) % state % thickness

Table B.2: thickness: ice thickness

B.1.3 layerThickness

Type:	real
Units:	*m*
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % layerThickness

Table B.3: layerThickness: layer thickness

B.1.4 temperature

Type:	real
Units:	*degrees Celsius*
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	Input Restart Output
Index in tracers Array:	domain % blocklist % state % index_temperature
Location in code:	domain % blocklist % state % time_levs(:) % state % tracers
Array Group:	dynamics

Table B.4: temperature: ice temperature

B.1.5 lowerSurface

Type:	real
Units:	*m above datum*
Dimension:	nCells Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % lowerSurface

Table B.5: lowerSurface: elevation at bottom of ice

B.1.6 upperSurface

Type:	real
Units:	*m above datum*
Dimension:	nCells Time

Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % upperSurface

Table B.6: upperSurface: elevation at top of ice

B.1.7 layerThicknessEdge

Type:	real
Units:	*m*
Dimension:	nVertLevels nEdges Time
Persistence:	persistent
Default Streams:	
Location in code:	domain % blocklist % state % time_levs(:) % state % layerThicknessEdge

Table B.7: layerThicknessEdge: layer thickness on cell edges

B.1.8 cellMask

Type:	integer
Units:	*none*
Dimension:	nCells Time
Persistence:	persistent
Default Streams:	Restart Output
Location in code:	domain % blocklist % state % time_levs(:) % state % cellMask

Table B.8: cellMask: bitmask indicating various properties about the ice sheet on cells. cellMask only needs to be a restart field if config_allow_additional_advance = false (to keep the mask of initial ice extent)

B.1.9 edgeMask

Type:	integer
Units:	*none*
Dimension:	nEdges Time
Persistence:	persistent
Default Streams:	Output

Location in code:	domain % blocklist % state % time_levs(:) % state % edgeMask

Table B.9: edgeMask: bitmask indicating various properties about the ice sheet on edges.

B.1.10 vertexMask

Type:	integer
Units:	*none*
Dimension:	nVertices Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % vertexMask

Table B.10: vertexMask: bitmask indicating various properties about the ice sheet on vertices.

B.1.11 normalVelocity

Type:	real
Units:	$m\ s^{-1}$
Dimension:	nVertLevels nEdges Time
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % state % time_levs(:) % state % normalVelocity

Table B.11: normalVelocity: horizonal velocity, normal component to an edge

B.1.12 uReconstructX

Type:	real
Units:	$m\ s^{-1}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % uReconstructX

Table B.12: uReconstructX: x-component of velocity reconstructed on cell centers

B.1.13 uReconstructY

Type:	real
Units:	$m\ s^{-1}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % uReconstructY

Table B.13: uReconstructY: y-component of velocity reconstructed on cell centers

B.1.14 uReconstructZ

Type:	real
Units:	$m\ s^{-1}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % uReconstructZ

Table B.14: uReconstructZ: z-component of velocity reconstructed on cell centers

B.1.15 uReconstructZonal

Type:	real
Units:	$m\ s^{-1}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % uReconstructZonal

Table B.15: uReconstructZonal: zonal velocity reconstructed on cell centers

B.1.16 uReconstructMeridional

Type:	real
Units:	$m\ s^{-1}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % uReconstructMeridional

Table B.16: uReconstructMeridional: meridional velocity reconstructed on cell centers

B.2 tend

B.2.1 tend_layerThickness

Type:	real
Units:	$m\ s^{-1}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % tend % layerThickness

Table B.17: tend_layerThickness: time tendency of layer thickness

B.2.2 tend_temperature

Type:	real
Units:	$K\ s^{-1}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	None
Index in tracers Array:	domain % blocklist % tend % index_temperature
Location in code:	domain % blocklist % tend % tracers
Array Group:	dynamics

Table B.18: tend_temperature: time tendency of ice temperature

B.3 mesh

B.3.1 latCell

Type:	real
Units:	*radians*
Dimension:	nCells
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % latCell

Table B.19: latCell: Latitude location of cell centers in radians.

B.3.2 lonCell

Type:	real
Units:	*radians*
Dimension:	nCells
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % lonCell

Table B.20: lonCell: Longitude location of cell centers in radians.

B.3.3 xCell

Type:	real
Units:	*unitless*
Dimension:	nCells
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % xCell

Table B.21: xCell: X Coordinate in cartesian space of cell centers.

B.3.4 yCell

Type:	real
Units:	*unitless*
Dimension:	nCells
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % yCell

Table B.22: yCell: Y Coordinate in cartesian space of cell centers.

B.3.5 zCell

Type:	real
Units:	*unitless*
Dimension:	nCells
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % zCell

Table B.23: zCell: Z Coordinate in cartesian space of cell centers.

B.3.6 indexToCellID

Type:	integer
Units:	*unitless*
Dimension:	nCells
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % indexToCellID

Table B.24: indexToCellID: List of global cell IDs.

B.3.7 latEdge

Type:	real
Units:	*radians*
Dimension:	nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % latEdge

Table B.25: latEdge: Latitude location of edge midpoints in radians.

B.3.8 lonEdge

Type:	real
Units:	*radians*
Dimension:	nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % lonEdge

Table B.26: lonEdge: Longitude location of edge midpoints in radians.

B.3.9 xEdge

Type:	real
Units:	*unitless*
Dimension:	nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % xEdge

Table B.27: xEdge: X Coordinate in cartesian space of edge midpoints.

B.3.10 yEdge

Type:	real
Units:	*unitless*
Dimension:	nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % yEdge

Table B.28: yEdge: Y Coordinate in cartesian space of edge midpoints.

B.3.11 zEdge

Type:	real
Units:	*unitless*
Dimension:	nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % zEdge

Table B.29: zEdge: Z Coordinate in cartesian space of edge midpoints.

B.3.12 indexToEdgeID

Type:	integer
Units:	*unitless*
Dimension:	nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % indexToEdgeID

Table B.30: indexToEdgeID: List of global edge IDs.

B.3.13 latVertex

Type:	real
Units:	*radians*
Dimension:	nVertices
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % latVertex

Table B.31: latVertex: Latitude location of vertices in radians.

B.3.14 lonVertex

Type:	real
Units:	*radians*
Dimension:	nVertices
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % lonVertex

Table B.32: lonVertex: Longitude location of vertices in radians.

B.3.15 xVertex

Type:	real
Units:	*unitless*
Dimension:	nVertices

Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % xVertex

Table B.33: xVertex: X Coordinate in cartesian space of vertices.

B.3.16 yVertex

Type:	real
Units:	*unitless*
Dimension:	nVertices
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % yVertex

Table B.34: yVertex: Y Coordinate in cartesian space of vertices.

B.3.17 zVertex

Type:	real
Units:	*unitless*
Dimension:	nVertices
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % zVertex

Table B.35: zVertex: Z Coordinate in cartesian space of vertices.

B.3.18 indexToVertexID

Type:	integer
Units:	*unitless*
Dimension:	nVertices
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % indexToVertexID

Table B.36: indexToVertexID: List of global vertex IDs.

B.3.19 cellsOnEdge

Type:	integer
Units:	*unitless*
Dimension:	TWO nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % cellsOnEdge

Table B.37: cellsOnEdge: List of cells that straddle each edge.

B.3.20 nEdgesOnCell

Type:	integer
Units:	*unitless*
Dimension:	nCells
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % nEdgesOnCell

Table B.38: nEdgesOnCell: Number of edges that border each cell.

B.3.21 nEdgesOnEdge

Type:	integer
Units:	*unitless*
Dimension:	nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % nEdgesOnEdge

Table B.39: nEdgesOnEdge: Number of edges that surround each of the cells that straddle each edge. These edges are used to reconstruct the tangential velocities.

B.3.22 edgesOnCell

Type:	integer
Units:	*unitless*
Dimension:	maxEdges nCells
Persistence:	persistent

Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % edgesOnCell

Table B.40: edgesOnCell: List of edges that border each cell.

B.3.23 edgesOnEdge

Type:	integer
Units:	*unitless*
Dimension:	maxEdges2 nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % edgesOnEdge

Table B.41: edgesOnEdge: List of edges that border each of the cells that straddle each edge.

B.3.24 weightsOnEdge

Type:	real
Units:	*unitless*
Dimension:	maxEdges2 nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % weightsOnEdge

Table B.42: weightsOnEdge: Reconstruction weights associated with each of the edgesOnEdge.

B.3.25 dvEdge

Type:	real
Units:	m
Dimension:	nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % dvEdge

Table B.43: dvEdge: Length of each edge, computed as the distance between verticesOnEdge.

B.3.26 dcEdge

Type:	real
Units:	m
Dimension:	nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % dcEdge

Table B.44: dcEdge: Length of each edge, computed as the distance between cell-sOnEdge.

B.3.27 angleEdge

Type:	real
Units:	$radians$
Dimension:	nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % angleEdge

Table B.45: angleEdge: Angle the edge normal makes with local eastward direction.

B.3.28 areaCell

Type:	real
Units:	m^2
Dimension:	nCells
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % areaCell

Table B.46: areaCell: Area of each cell in the primary grid.

B.3.29 areaTriangle

Type:	real
Units:	m^2

Dimension:	nVertices
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % areaTriangle

Table B.47: areaTriangle: Area of each cell (triangle) in the dual grid.

B.3.30 edgeNormalVectors

Type:	real
Units:	*unitless*
Dimension:	R3 nEdges
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % mesh % edgeNormalVectors

Table B.48: edgeNormalVectors: Normal vector defined at an edge.

B.3.31 localVerticalUnitVectors

Type:	real
Units:	*unitless*
Dimension:	R3 nCells
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % mesh % localVerticalUnitVectors

Table B.49: localVerticalUnitVectors: Unit surface normal vectors defined at cell centers.

B.3.32 cellTangentPlane

Type:	real
Units:	*unitless*
Dimension:	R3 TWO nCells
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % mesh % cellTangentPlane

Table B.50: cellTangentPlane: The two vectors that define a tangent plane at a cell center.

B.3.33 cellsOnCell

Type:	integer
Units:	*unitless*
Dimension:	maxEdges nCells
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % cellsOnCell

Table B.51: cellsOnCell: List of cells that neighbor each cell.

B.3.34 verticesOnCell

Type:	integer
Units:	*unitless*
Dimension:	maxEdges nCells
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % verticesOnCell

Table B.52: verticesOnCell: List of vertices that border each cell.

B.3.35 verticesOnEdge

Type:	integer
Units:	*unitless*
Dimension:	TWO nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % verticesOnEdge

Table B.53: verticesOnEdge: List of vertices that straddle each edge.

B.3.36 edgesOnVertex

Type:	integer
Units:	*unitless*
Dimension:	vertexDegree nVertices

Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % edgesOnVertex

Table B.54: edgesOnVertex: List of edges that share a vertex as an endpoint.

B.3.37 cellsOnVertex

Type:	integer
Units:	*unitless*
Dimension:	vertexDegree nVertices
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % cellsOnVertex

Table B.55: cellsOnVertex: List of cells that share a vertex.

B.3.38 kiteAreasOnVertex

Type:	real
Units:	m^2
Dimension:	vertexDegree nVertices
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % kiteAreasOnVertex

Table B.56: kiteAreasOnVertex: Area of the portions of each dual cell that are part of each cellsOnVertex.

B.3.39 coeffs_reconstruct

Type:	real
Units:	*unitless*
Dimension:	R3 maxEdges nCells
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % mesh % coeffs_reconstruct

Table B.57: coeffs_reconstruct: Coefficients to reconstruct velocity vectors at cells centers.

B.3.40 edgeSignOnCell

Type:	integer
Units:	*unitless*
Dimension:	maxEdges nCells
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % mesh % edgeSignOnCell

Table B.58: edgeSignOnCell: Sign of edge contributions to a cell for each edge on cell. Used for bit-reproducible loops. Represents directionality of vector connecting cells.

B.3.41 edgeSignOnVertex

Type:	integer
Units:	*unitless*
Dimension:	maxEdges nVertices
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % mesh % edgeSignOnVertex

Table B.59: edgeSignOnVertex: Sign of edge contributions to a vertex for each edge on vertex. Used for bit-reproducible loops. Represents directionality of vector connecting vertices.

B.3.42 layerThicknessFractions

Type:	real
Units:	*none*
Dimension:	nVertLevels
Persistence:	persistent
Default Streams:	Input Restart
Location in code:	domain % blocklist % mesh % layerThicknessFractions

Table B.60: layerThicknessFractions: Fractional thickness of each sigma layer

B.3.43 layerCenterSigma

Type:	real
Units:	*none*
Dimension:	nVertLevels
Persistence:	persistent
Default Streams:	
Location in code:	domain % blocklist % mesh % layerCenterSigma

Table B.61: layerCenterSigma: Sigma (fractional) level at center of each layer

B.3.44 layerInterfaceSigma

Type:	real
Units:	*none*
Dimension:	nVertLevelsP1
Persistence:	persistent
Default Streams:	
Location in code:	domain % blocklist % mesh % layerInterfaceSigma

Table B.62: layerInterfaceSigma: Sigma (fractional) level at interface between each layer (including top and bottom)

B.3.45 bedTopography

Type:	real
Units:	*m above datum*
Dimension:	nCells
Persistence:	persistent
Default Streams:	Input Restart
Location in code:	domain % blocklist % mesh % bedTopography

Table B.63: bedTopography: Elevation of ice sheet bed. Once isostasy is added to the model, this should become a state variable.

B.3.46 sfcMassBal

Type:	real
Units:	$kg\ m^2\ s^{-1}$
Dimension:	nCells
Persistence:	persistent
Default Streams:	Input Restart

Location in code:	domain % blocklist % mesh % sfcMassBal

Table B.64: sfcMassBal: Surface mass balance